Statistical Techniques in Robotics (16-831, S20) Lecture #16 (Wednesday, May 31)

Model Based Reinforcement Learning

Lecturer: Kris Kitani *Scribes: Jeet Kanjani, Shubham Gupta*

1 Review

Markov Decision Process

A Markov Decision Process is a mathematical framework for describing a fully observable environment where the outcomes are partly random and partly under control of the agent. [1] At every timestep, the agent samples an action from policy (distribution) $\pi(a|x)$, and uses the state transition probability $p(s'|s,a)$ to determine the next state s'. The stochastic nature of the environment causes the agent to be in different states given the current state and action. The reward agent recieves at every timestep is denoted by $r(s',s,a)$. A markov decision process can be represented as a graphical model:

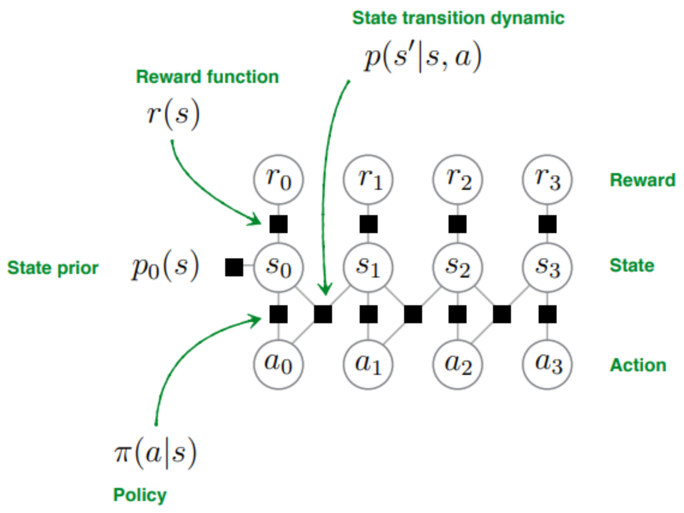


Figure 1: Markov Decision Process

This factorization of this model can be written as:

$$p(s_0, s_1, a_1, \cdots, s_T) = p_0(s_0) \prod_t p(s_{t+1} \mid s_t, a_t)\, p(a_t \mid s_t) \tag{1}$$

In RL, we try to maximize the cumulative reward over a sequence of states and actions. In model based RL, we have access to the $p(s'|s,a)$ and the reward function which we can use to learn a policy $\pi(a|x)$ that maximizes the cumulative reward.

Value Function

The value function represents how good is a state for an agent to be in. It is equal to the expected total reward for the agent starting from state s. The value function depends on the policy by which the agent picks actions to perform.[2] The state-action value is given by:

$$Q^{\pi}(s,a)=\mathbb{E}_p\left[\gamma^0 r\left(s_0\right)+\gamma^1 r\left(s_1\right)+\gamma^2 r\left(s_2\right)+\cdots \mid s_0=s, a_0=a\right]$$

The relationship between the state value function and state-action value function is given by:

$$V^{\pi}(s)=\sum_a \pi(a \mid s) Q^{\pi}(s,a)$$

2 Summary

2.1 Bellman Optimality Equation

The state-value function is a linear combination of the immediate reward r_t plus the discounted reward of the next state. This results in a recursive relationship between state value functions for a given policy:

$$V^{\pi}(s)=\mathbb{E}_p\left[r_t+\gamma V^{\pi}\left(s_{t+1}\right) \mid s_t=s\right]$$

This can be interpreted as future rewards i.e. value obtained after taking an action and receiving a reward. The relationship between state-action value functions that can be obtained is known as the Bellman equation.

$$Q^{\pi}(s,a)=\mathbb{E}_p\left[r_t+\gamma Q^{\pi}\left(s_{t+1}, a_{t+1}\right) \mid s_t=s, a_t=a\right]$$

This gives us the methods to derive the Bellman optimality equations. When comparing two policies, the policy with the better value function should give a better return.

$$V^{\pi^*}(s)=\max_{\pi} V^{\pi}(s)$$
$$Q^{\pi^*}(s,a)=\max_{\pi} Q^{\pi}(s,a)$$

We get a recursive relation when we substitute the expectation into the previous expectation. This is the Bellman Optimality equation:

$$\begin{gathered}V^{\pi^*}(s)=\max_a \textstyle\sum_{s'} p\left(s' \mid s,a\right)\left[r_t+\gamma V^{\pi^*}\left(s'\right)\right] \\ Q^{\pi^*}(s,a)=\textstyle\sum_{s'} p\left(s' \mid s,a\right)\left[r(s)+\gamma \max_{a'} Q^{\pi^*}\left(s', a'\right)\right]\end{gathered}$$

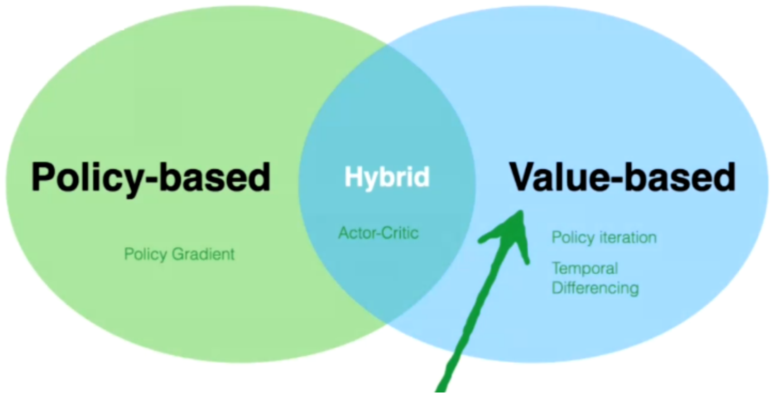
The proof of the above equation is given in the appendix.

Computing the optimal policy

The goal of reinforcement learning is to learn the policy which maximizes the expected return.

$$\hat{\pi} = \arg\max_{\pi} \mathbb{E}_{p_\pi(\zeta)} \left[\sum_{t=0}^{T} r^{(t)} \right] \tag{2}$$

Since the policy can be derived from the state-action value function, we just need the value function to obtain the optimal policy. This is called value-based reinforcement learning where we iteratively try to find the optimal value function which would entail an optimal policy.



$$\pi(s) = \arg\max_{a} Q(s, a) \tag{3}$$

In RL, the task of **Prediction** is referred to as the task to solve for the optimal policy.

There are two types of prediction methods depending on what is assumed for the environment.

- **Model Based**: Model is fully specified (Access to inner working of the environment)
- **Model-free** Model is limited to interactions

2.2 RL Approaches for Optimal Policy

The various approaches to solve RL can be divided in 3 categories:

- **Policy-based:** These methods build an explicit policy representation. Eg - policy gradients
- **Value-based:** These methods use value functions to derive the optimal policy (implicit representation). Eg- policy iteration, temporal differencing
- **Hybrid:** Mix of above 2. Eg- actor critic

We will solve the value-based model based RL in the next section.

2.3 Value-based Model-based RL

2.3.1 Chicken and Egg Problem

In order to find the optimal policy using the value function, we can use the Bellman optimality equation.
Let's formalize this mathematically. We have

- Finite state space $s \in S$
- Finite action space $a \in A$
- State transition dynamics $p(s^{'}|s, a)$
- Reward function $r(s^{'}, a, s)$

We need to estimate the value function $V^{\pi}(s)$ or $Q^{\pi}(s, a)$ which in turn can be used to find the optimal policy π^{*} using the Bellman optimality equation. Problem is that in order to estimate the value function we also need the policy $\pi(a|s)$. Now, this becomes like a chicken and egg problem where we want to find the optimal policy using the value function but the value function estimation also needs policy.

2.3.2 Policy Iteration

The above-stated chicken and egg problem can be solved iteratively. We can start with an initial guess for policy, compute the value function for this policy using the Bellman equation, and use the Bellman optimality equation to find a new optimal policy for this value function. We can keep on doing this iteratively till convergence is met.

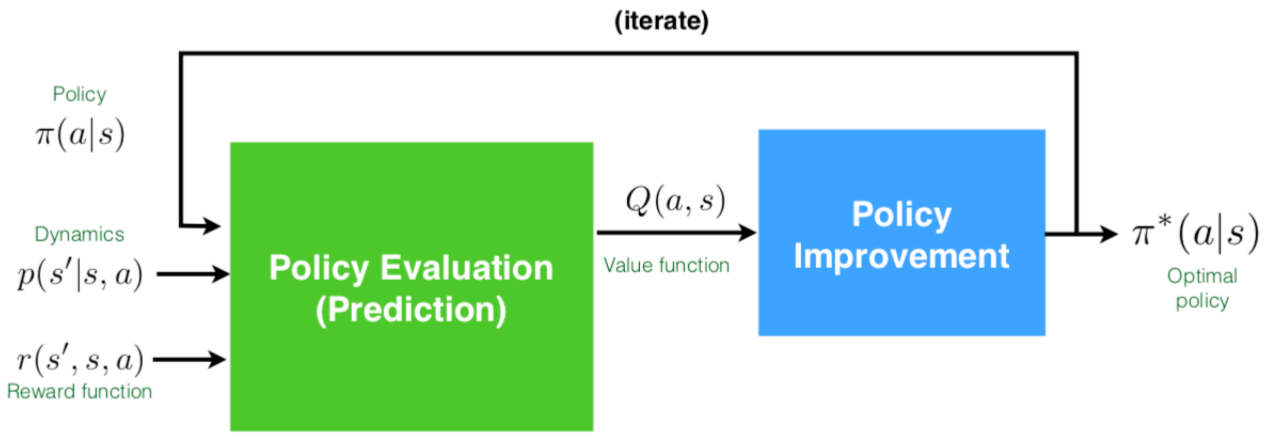


Figure 2: Policy Iteration Method for RL

Here the step to estimate the value function is called the Policy Evaluation (Prediction). The estimation of the optimal policy is called Policy Improvement.

2.3.3 Policy Evaluation a.k.a Prediction

The prediction block is the most important block in model-based RL methods. This block takes as input the current policy, state transition dynamics, and reward function and derives the optimal value function by recursion. Since it involves multiple iterations over all the possible action and state space, it is a highly computationally expensive block.

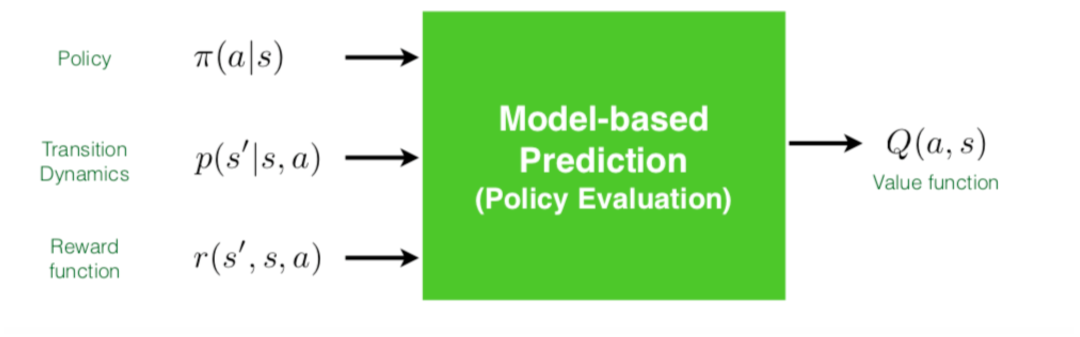


Figure 3: Policy Evaluation Block

2.3.4 Policy Improvement

Given a value function, the policy improvement block finds the optimal policy using the Bellman optimality equation. This can be simply computed by finding the optimal action for each of the possible states using the value function.

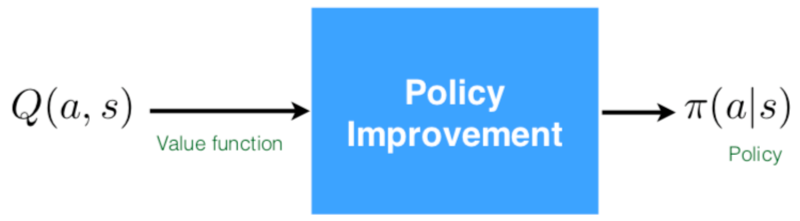


Figure 4: Policy Improvement Block

2.3.5 Algorithm

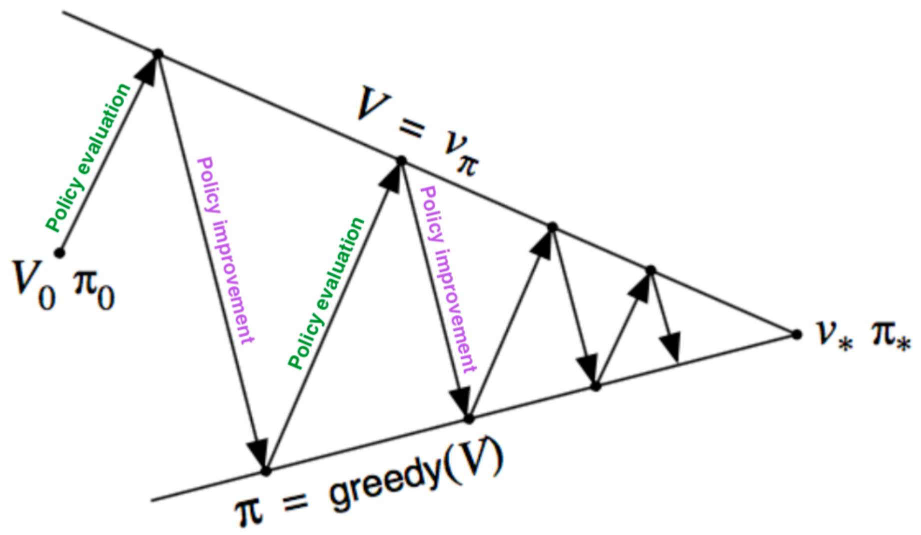
We can formalize the above approach in the form of a pseudo-code mentioned below.

Algorithm 1 function POLICYITERATION $(r(s), p(s' \mid s, a), \gamma)$

```
 1: $\pi \leftarrow \text{rand}(\mathcal{A})$
 2: $V \leftarrow \text{rand}(\mathbb{R})$
 3: $V' \leftarrow \text{rand}(\mathbb{R})$
 4: while $\max_s |V(s) - V'(s)| \geq \epsilon$ do
 5:     $V' \leftarrow V$
 6:     for $s \in \mathcal{S}$ do
 7:         $Q(s,a) = r(s) + \gamma \sum_{s'} p(s' \mid s, a) V'(s') \quad \forall a$
 8:         $V(s) = \sum_a \pi(a \mid s) Q(s,a)$
 9:     end for
10: end while
11: for $s \in \mathcal{S}$ do
12:     $\pi'(s) = \arg\max_a Q(s,a)$
13: end for
14: if $\pi' = \pi$ then
15:     return $\pi$
16: end if
17: Go to line 2
```

2.4 Computational Optimization

We can represent the above algorithm graphically as shown below.



Here, we can observe that we need to perform multiple iterations to compute the value function at each given policy before updating the current policy. Instead of doing this, we can actually keep on updating the policy each time we estimate a new value function. This can be thought of as applying stochastic gradient descent in place of batch gradient descent for faster convergence.

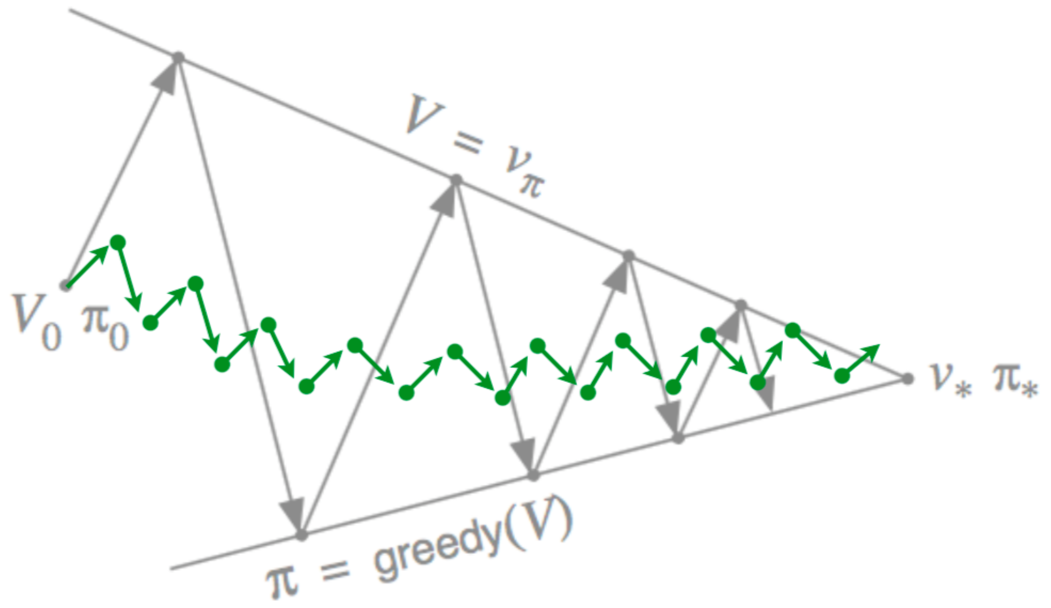
2.4.1 Value iteration

The above-mentioned idea of simultaneously updating the policy and value function is called value iteration.

Algorithm 2 function VALUEITERATION $(r(s), p(s' \mid s, a), \gamma)$

```
1:  $\pi \leftarrow \mathrm{rand}(\mathcal{A})$
2:  $V \leftarrow \mathrm{rand}(\mathbb{R})$
3:  $V' \leftarrow \mathrm{rand}(\mathbb{R})$
4:  while $\max_s |V(s) - V'(s)| \geq \epsilon$ do
5:      $V' \leftarrow V$
6:      for $s \in \mathcal{S}$ do
7:          $Q(s,a) = r(s) + \gamma \sum_{s'} p(s' \mid s, a) V'(s') \quad \forall a$
8:          $\pi'(s) = \arg\max_a Q(s,a)$
9:          $V(s) = \sum_a \pi(a \mid s) Q(s,a)$
10:     end for
11: end while
12: if $\pi' = \pi$ then
13:     return $\pi$
14: end if
15: Go to line 2
```

As shown in the below figure, the value iteration method converges much faster in comparison to the sequential update method.



References

[1] F Concina Reinforcement Learning - Markov Decision Process
[Markov Decision Process](https://fabioconcina.github.io/blog/markov-decision-process)

[2] Deep Reinforcement Learning Demystified - Policy Iteration, Value Iteration and Q learning
https://medium.com/@m.alzantot/deep-reinforcement-learning-demysitifed-episode-2-policy-iteration-value-iteration-and-q-978f9e89ddaa

2.5 Appendix

Proof for Bellman Optimality equation:

$$\begin{aligned}
V^{\pi^*}(s) &= \sum_a \pi(a \mid s) Q^{\pi^*}(s,a) \\
&= \max_a Q^{\pi^*}(s,a) \\
&= \max_a \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_t \mid s_0 = s, a_0 = a\right] \\
&= \max_a \mathbb{E}\left[r_0 + \sum_{t=1}^{\infty} \gamma^t r_t \mid s_0 = s, a_0 = a\right] \\
&= \max_a \sum_{s'} p\left(s_1 = s' \mid s, a\right)\left[r_0 + \mathbb{E}\left\{\sum_{t=1}^{\infty} \gamma^t r_t \mid s_1 = s'\right\}\right]
\end{aligned} \tag{4}$$